from flask import Flask, request, jsonify

from flask\_cors import CORS

import difflib

import os

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from werkzeug.utils import secure\_filename

from PyPDF2 import PdfReader

import docx

from transformers import pipeline

app = Flask(\_\_name\_\_)

CORS(app)

# Allowed file types

ALLOWED\_EXTENSIONS = {'txt', 'pdf', 'docx'}

UPLOAD\_FOLDER = 'uploads'

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

# Store assignments in a dictionary

assignments = {}

# AI detection model

try:

ai\_detector = pipeline("text-classification", model="roberta-base-openai-detector", framework="pt")

print("AI Detection Model Loaded Successfully.")

except Exception as e:

ai\_detector = None

print(f"AI model could not be loaded: {e}")

# Check if a file is allowed

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

# Extract text from different file formats

def extract\_text\_from\_file(file\_path):

file\_extension = os.path.splitext(file\_path)[1].lower()

try:

if file\_extension == '.txt':

with open(file\_path, 'r', encoding='utf-8', errors='ignore') as file:

return file.read()

elif file\_extension == '.pdf':

reader = PdfReader(file\_path)

return ''.join([page.extract\_text() or '' for page in reader.pages])

elif file\_extension == '.docx':

doc = docx.Document(file\_path)

return '\n'.join([para.text for para in doc.paragraphs])

except Exception as e:

print(f"Error reading {file\_path}: {e}")

return ''

# Compute cosine similarity between assignments

def compute\_similarity():

if len(assignments) < 2:

return np.array([]) # Return empty if there are not enough assignments

texts = list(assignments.values())

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(texts)

return cosine\_similarity(tfidf\_matrix)

# Highlight differences between two texts

def highlight\_differences(text1, text2, sim\_score):

if not text1 or not text2:

return "No text to compare"

text1\_clean = text1.split()

text2\_clean = text2.split()

diff = list(difflib.ndiff(text1\_clean, text2\_clean))

highlighted\_text = []

for word in diff:

if word.startswith("+ "): # Added in second text

highlighted\_text.append(f'\*\*{word[2:]}\*\*')

elif word.startswith("- "): # Removed from second text

highlighted\_text.append(f'~~{word[2:]}~~')

else:

highlighted\_text.append(word[2:])

status = "✅ Unique"

if sim\_score >= 80:

status = "❌ Copied"

elif sim\_score >= 50:

status = "⚠️ Partially Copied"

return f"{status}: {' '.join(highlighted\_text)}"

# Route to upload files

@app.route('/upload', methods=['POST'])

def upload\_assignments():

if 'files' not in request.files:

return jsonify({"error": "No file part in the request"}), 400

files = request.files.getlist('files')

if not files:

return jsonify({"error": "No files selected"}), 400

assignments.clear()

for file in files:

if file.filename == '':

return jsonify({"error": "One of the files has no filename"}), 400

if file and allowed\_file(file.filename):

filename = secure\_filename(file.filename)

file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(file\_path)

student\_name = filename.rsplit('.', 1)[0]

count = 1

while student\_name in assignments:

student\_name = f"{filename.rsplit('.', 1)[0]}\_{count}"

count += 1

file\_content = extract\_text\_from\_file(file\_path)

if file\_content.strip():

assignments[student\_name] = file\_content

else:

return jsonify({"error": f"Could not extract text from '{filename}'"}), 400

else:

return jsonify({"error": f"File '{file.filename}' is not a valid type."}), 400

print(f"Uploaded {len(files)} files. Current assignments: {list(assignments.keys())}")

return jsonify({"message": f"{len(files)} files uploaded successfully!"})

# Route to compare assignments

@app.route('/compare', methods=['GET'])

def compare\_assignments():

if len(assignments) < 2:

return jsonify({"error": "Not enough assignments to compare"}), 400

similarity\_matrix = compute\_similarity()

if similarity\_matrix.size == 0:

return jsonify([])

students = list(assignments.keys())

results = []

ai\_results = {}

summary\_data = {

"total\_comparisons": 0,

"copied\_count": 0,

"partially\_copied\_count": 0,

"ai\_generated\_count": 0,

"ai\_students": []

}

for i in range(len(students)):

for j in range(i + 1, len(students)):

sim\_score = similarity\_matrix[i][j] \* 100

highlighted\_text = highlight\_differences(assignments[students[i]], assignments[students[j]], sim\_score)

results.append({

"student1": students[i],

"student2": students[j],

"similarity": round(sim\_score, 2),

"highlighted": highlighted\_text

})

# Update summary counts

if sim\_score >= 80:

summary\_data["copied\_count"] += 1

elif sim\_score >= 50:

summary\_data["partially\_copied\_count"] += 1

summary\_data["total\_comparisons"] += 1

if ai\_detector:

try:

ai\_score = ai\_detector(assignments[students[i]])[0]

ai\_results[students[i]] = {

"label": ai\_score['label'],

"score": round(ai\_score['score'] \* 100, 2)

}

# If AI detected, add to summary

if ai\_score['label'] == 'LABEL\_1': # Assuming LABEL\_1 corresponds to AI-generated

summary\_data["ai\_generated\_count"] += 1

summary\_data["ai\_students"].append({

"student": students[i],

"ai\_score": ai\_results[students[i]]["score"]

})

except Exception as e:

ai\_results[students[i]] = {"error": f"AI detection failed: {str(e)}"}

combined\_results = []

for res in results:

student1 = res["student1"]

student2 = res["student2"]

plagiarism\_result = res

ai\_result\_student1 = ai\_results.get(student1, {"label": "Unknown", "score": 0})

ai\_result\_student2 = ai\_results.get(student2, {"label": "Unknown", "score": 0})

combined\_results.append({

"student1": student1,

"student2": student2,

"similarity": plagiarism\_result["similarity"],

"highlighted": plagiarism\_result["highlighted"],

"ai\_detection\_student1": ai\_result\_student1,

"ai\_detection\_student2": ai\_result\_student2

})

return jsonify({

"comparisons": combined\_results,

"summary": summary\_data

})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)